What's New in HTTP Live Streaming

Session 504

Roger Pantos Media Systems Engineer
Jordan Schneider Media Systems Engineer
HTTP Live Streaming in 20 Seconds

#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
HTTP Live Streaming in 20 Seconds

#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"

#EXTINF 10.001
segment1.ts

#EXTINF 10.001
segment2.ts
HTTP Live Streaming in 20 Seconds

#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
HTTP Live Streaming in 20 Seconds

#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
HTTP Live Streaming in 20 Seconds

#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
HTTP Live Streaming in 20 Seconds

#EXTM3U

#EXT-X-VERSION:4

#EXT-X-TARGETDURATION:10

#EXT-X-MEDIA-SEQUENCE:1

#EXT-X-MAP:URI="patpmt.ts"

#EXTINF 10.001

segment1.ts

#EXTINF 10.001

segment2.ts

#EXTINF 10.001

segment3.ts
MPEG-4 Fragment Support in HLS
MPEG-4 Fragment Support in HLS

Extension of the familiar MPEG-4 file format (myMovie.mp4)

• An MPEG-4 file has a sample table followed by sample data
MPEG-4 Fragment Support in HLS

Extension of the familiar MPEG-4 file format (myMovie.mp4)

• An MPEG-4 file has a sample table followed by sample data

fMP4 “Fragments” divide myMovie.mp4 into separately decodable chunks

• Each with its own sample table and sample data
MPEG-4 Fragment Support in HLS

Extension of the familiar MPEG-4 file format (myMovie.mp4)

• An MPEG-4 file has a sample table followed by sample data

fMP4 “Fragments” divide myMovie.mp4 into separately decodable chunks

• Each with its own sample table and sample data

Adding fragmented MP4 as a supported Segment format to HLS spec

• Beta version available to Apple Developer Program members
MPEG-4 Fragment Support in HLS

Extension of the familiar MPEG-4 file format (myMovie.mp4)
- An MPEG-4 file has a sample table followed by sample data
fMP4 “Fragments” divide myMovie.mp4 into separately decodable chunks
- Each with its own sample table and sample data

Adding fragmented MP4 as a supported Segment format to HLS spec
- Beta version available to Apple Developer Program members
fMP4 Segments support the same set of features as TS
MPEG-4 Fragment Support in HLS

Extension of the familiar MPEG-4 file format (myMovie.mp4)

- An MPEG-4 file has a sample table followed by sample data
- fMP4 “Fragments” divide myMovie.mp4 into separately decodable chunks
- Each with its own sample table and sample data

Adding fragmented MP4 as a supported Segment format to HLS spec

- Beta version available to Apple Developer Program members

fMP4 Segments support the same set of features as TS

Works on iOS, macOS, and tvOS
Benefits of fMP4 Segments

Allows a single media library to be delivered to multiple ecosystems
- HLS, MPEG-DASH, others
- Increases CDN cache efficiency
Benefits of fMP4 Segments

Allows a single media library to be delivered to multiple ecosystems
- HLS, MPEG-DASH, others
- Increases CDN cache efficiency

Common authoring and validation tools across ecosystems
Benefits of fMP4 Segments

Allows a single media library to be delivered to multiple ecosystems
• HLS, MPEG-DASH, others
• Increases CDN cache efficiency
Common authoring and validation tools across ecosystems
Higher network efficiency at low bit rates
How Does HLS Change?

#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="patpmt.ts"
#EXTINF 10.001
segment1.ts
#EXTINF 10.001
segment2.ts
How Does HLS Change?

#EXTM3U
#EXT-X-VERSION:4
#EXT-X-TARGETDURATION:10
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-MAP:URI="moov.mp4"
#EXTINF 10.001
  segment1.mp4
#EXTINF 10.001
  segment2.mp4
Encrypting fMP4 Segments

Whole-segment encryption is same as TS
Encrypting fMP4 Segments

Whole-segment encryption is same as TS

Sample encryption uses part of ISO/IEC 23001:7 2016

- MPEG standard—“Common Encryption”
- 'cbcs' mode
Getting to Interoperability

Achieving a single media library

MPEG is working to define a “Common Media Application Format” (CMAF)

• Originally proposed by Apple and Microsoft

• Has attracted broad support at MPEG
Getting to Interoperability

Achieving a single media library

MPEG is working to define a “Common Media Application Format” (CMAF)

• Originally proposed by Apple and Microsoft
• Has attracted broad support at MPEG

Constrains the MPEG-4 Fragment definition (ISO 14496 Part 12)

• Requires unmixed audio and video delivery
• Requires that every video segment start with a key frame
• Requires precise segment alignment across bit rate variants
• And more
In-Playlist Timed Metadata
# A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
</table>

NEW
### A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td></td>
</tr>
</tbody>
</table>
## A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>Usually authored as text</td>
</tr>
</tbody>
</table>
# A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td></td>
</tr>
<tr>
<td>Usually authored as text</td>
<td></td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td></td>
</tr>
</tbody>
</table>
A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>Usually authored as text</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Easily added to playlist or JSON</td>
</tr>
<tr>
<td>Static</td>
<td>Static</td>
</tr>
</tbody>
</table>
### A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td></td>
</tr>
<tr>
<td>Usually authored as text</td>
<td></td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td></td>
</tr>
<tr>
<td>Static</td>
<td></td>
</tr>
<tr>
<td>Available immediately</td>
<td></td>
</tr>
</tbody>
</table>
A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td></td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td></td>
</tr>
<tr>
<td>Static</td>
<td></td>
</tr>
<tr>
<td>Available immediately</td>
<td></td>
</tr>
</tbody>
</table>
# A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td></td>
</tr>
<tr>
<td>Static</td>
<td></td>
</tr>
<tr>
<td>Available immediately</td>
<td></td>
</tr>
</tbody>
</table>
# A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td>Available immediately</td>
</tr>
</tbody>
</table>

- *Static metadata* are usually authored as text and can be easily added to playlist or JSON.
- *ID3 timed metadata* are in a binary format (ID3) and require specialized tools.
# A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Available immediately</td>
<td></td>
</tr>
</tbody>
</table>
# A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Available immediately</td>
<td>Delivered as played</td>
</tr>
</tbody>
</table>
# A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Available immediately</td>
<td>Delivered as played</td>
</tr>
</tbody>
</table>

- **Static metadata**
  - e.g. content title
  - Usually authored as text
  - Easily added to playlist or JSON
  - Static
  - Available immediately

- **ID3 timed metadata**
  - e.g. ad marker
  - Binary format (ID3)
  - Requires specialized tools
  - Dynamic
  - Delivered as played
## A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>In-playlist timed metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>-</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>-</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td>-</td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td>-</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Available immediately</td>
<td>-</td>
<td>Delivered as played</td>
</tr>
</tbody>
</table>

- **Static metadata**
  - Usually authored as text
  - Easily added to playlist or JSON
  - Available immediately

- **In-playlist timed metadata**
  - Delivered as played

- **ID3 timed metadata**
  - Binary format (ID3)
  - Requires specialized tools
  - Delivered as played
## A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>In-playlist timed metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td></td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td></td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td></td>
<td>Dynamic</td>
</tr>
<tr>
<td>Available immediately</td>
<td></td>
<td>Delivered as played</td>
</tr>
</tbody>
</table>
## A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>In-playlist timed metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Usually authored as text</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td></td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td></td>
<td>Dynamic</td>
</tr>
<tr>
<td>Available immediately</td>
<td></td>
<td>Delivered as played</td>
</tr>
</tbody>
</table>
## A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>In-playlist timed metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Usually authored as text</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td>Easily added to playlist</td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td></td>
<td>Dynamic</td>
</tr>
<tr>
<td>Available immediately</td>
<td></td>
<td>Delivered as played</td>
</tr>
</tbody>
</table>
# A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>In-playlist timed metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Usually authored as text</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td>Easily added to playlist</td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td>Dynamic</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Available immediately</td>
<td></td>
<td>Delivered as played</td>
</tr>
</tbody>
</table>
# A Comparison of HLS Metadata

<table>
<thead>
<tr>
<th>Static metadata</th>
<th>In-playlist timed metadata</th>
<th>ID3 timed metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>e.g. content title</td>
<td>e.g. ad marker</td>
<td>e.g. ad marker</td>
</tr>
<tr>
<td>Usually authored as text</td>
<td>Usually authored as text</td>
<td>Binary format (ID3)</td>
</tr>
<tr>
<td>Easily added to playlist or JSON</td>
<td>Easily added to playlist</td>
<td>Requires specialized tools</td>
</tr>
<tr>
<td>Static</td>
<td>Dynamic</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Available immediately</td>
<td>Available immediately</td>
<td>Delivered as played</td>
</tr>
</tbody>
</table>
In-Playlist Timed Metadata
In-Playlist Timed Metadata

Metadata is expressed as a date-based range inside a playlist.
In-Playlist Timed Metadata

Metadata is expressed as a date-based range inside a playlist
Each range carries a content-defined set of attribute/value pairs
In-Playlist Timed Metadata

Metadata is expressed as a date-based range inside a playlist.
Each range carries a content-defined set of attribute/value pairs.
Ranges can be added and removed from live streams.
In-Playlist Timed Metadata

#EXT-X-DATERANGE

#EXTM3U


#EXTINF 10,
ad3.1.ts
#EXTINF 10,
ad3.2.ts
In-Playlist Timed Metadata

#EXT-X-DATERANGE

#EXTM3U


#EXTINF 10,
ad3.1.ts

#EXTINF 10,
ad3.2.ts
In-Playlist Timed Metadata
#EXT-X-DATERANGE

#EXTM3U

#EXTINF 10,
ad3.1.ts
#EXTINF 10,
ad3.2.ts
In-Playlist Timed Metadata

Content authoring
In-Playlist Timed Metadata

Content authoring

The DATERANGE tag can appear in both live and VOD playlists.
In-Playlist Timed Metadata

Content authoring

The DATERANGE tag can appear in both live and VOD playlists
Can be authored with media, or added in post-production
In-Playlist Timed Metadata

Content authoring

The DATERANGE tag can appear in both live and VOD playlists
Can be authored with media, or added in post-production
Spec includes bindings for SCTE-35 tags
In-Playlist Timed Metadata

Content authoring

The DATERANGE tag can appear in both live and VOD playlists
Can be authored with media, or added in post-production
Spec includes bindings for SCTE-35 tags
mediastreamvalidator support
In-Playlist Timed Metadata

Playback
In-Playlist Timed Metadata

Playback

AVFoundation API for obtaining DATE-RANGE info
In-Playlist Timed Metadata

Playback

AVFoundation API for obtaining DATE-RANGE info
All timed metadata available as soon as playlist is loaded
In-Playlist Timed Metadata

Playback

AVFoundation API for obtaining DATE-RANGE info

All timed metadata available as soon as playlist is loaded

Notification when list changes
let asset = AVURLAsset(url: url)
let playerItem = AVPlayerItem(asset: asset)
let collector = AVPlayerItemMetadataCollector()
collector.set(delegate: self, queue: mainQueue)
playerItem.addMediaDataCollector(collector)
Offline HLS Playback

Jordan Schneider Media Systems Engineer
What is Offline HLS?
What is Offline HLS?

HLS without network connectivity
What is Offline HLS?

HLS without network connectivity
Uses your existing media library
What is Offline HLS?

HLS without network connectivity
Uses your existing media library
Offline FairPlay Streaming
What is Offline HLS?

HLS without network connectivity
Uses your existing media library
Offline FairPlay Streaming
Downloads in the background
What is Offline HLS?

HLS without network connectivity
Uses your existing media library
Offline FairPlay Streaming
Downloads in the background
Plays while download is in progress
Should You Use Offline HLS?
Should You Use Offline HLS?
Advantages of Offline HLS
Advantages of Offline HLS
Advantages of Offline HLS

- Video Track
- English Audio
- Spanish Audio
- French Audio
- Chinese Audio
- English Commentary
Advantages of Offline HLS
Advantages of Offline HLS

- Video Track (video-hi.m3u8, video-low.m3u8)
- English Audio (en.m3u8)
- Spanish Audio (es.m3u8)
- French Audio (fr.m3u8)
- Chinese Audio (cn.m3u8)
- English Commentary (en-c.m3u8)
- English Subtitles (en.m3u8)
- Spanish Subtitles (es.m3u8)
- French Subtitles (fr.m3u8)
- Chinese Subtitles (cn.m3u8)
Advantages of Offline HLS

- Video Track: <video-hi.m3u8>, <video-low.m3u8>
- English Audio: <en.m3u8>
- Spanish Audio: <es.m3u8>
- French Audio: <fr.m3u8>
- Chinese Audio: <cn.m3u8>
- English Commentary: <en-c.m3u8>
- Spanish Subtitles: <es.m3u8>
- French Subtitles: <fr.m3u8>
- Chinese Subtitles: <cn.m3u8>
Advantages of Offline HLS

- Video Track
  - Video-high.m3u8
  - Video-low.m3u8

- Audio Tracks
  - English Audio
    - en.m3u8
  - Spanish Audio
    - es.m3u8
  - French Audio
    - fr.m3u8
  - Chinese Audio
    - cn.m3u8

- Commentary
  - English Commentary
    - en-c.m3u8

Downloaded

Not Downloaded
Advantages of Offline HLS
AVAssetDownloadTask
AVAssetDownloadTask

Inherits features of URLSession
Inherits features of URLSession

- Background downloading
AVAssetDownloadTask

Inherits features of URLSession

- Background downloading

Media selection
AVAssetDownloadTask

Inherits features of URLSession

- Background downloading

Media selection

Quality selection
public class AVAssetDownloadTask: URLSessionTask {
    ...
}

public class AVAssetDownloadURLSession: URLSession {
    func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String,
                                assetArtworkData artworkData: Data?,
                                options: [String : AnyObject]? = [:])
        -> AVAssetDownloadTask?
}

public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String
public let AVAssetDownloadTaskMediaSelectionKey: String
public class AVAssetDownloadTask: URLSessionTask {
    ...
}

public class AVAssetDownloadURLSession: URLSession {
    func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String,
                                assetArtworkData artworkData: Data?,
                                options: [String : AnyObject]? = [:])
        -> AVAssetDownloadTask?
}

public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String
public let AVAssetDownloadTaskMediaSelectionKey: String
public class AVAssetDownloadTask: URLSessionTask {
    ...
}

public class AVAssetDownloadURLSession: URLSession {
    func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String, assetArtworkData artworkData: Data?, options: [String : AnyObject]? = [:]) -> AVAssetDownloadTask?
}

public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String
public let AVAssetDownloadTaskMediaSelectionKey: String
public class AVAssetDownloadTask: URLSessionTask {

    ...

}

public class AVAssetDownloadURLSession: URLSession {

    func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String,
        assetArtworkData artworkData: Data?,
        options: [String : AnyObject]? = [:])
            -> AVAssetDownloadTask?

}

public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String

public let AVAssetDownloadTaskMediaSelectionKey: String
public class AVAssetDownloadTask: URLSessionTask {
    ...
}

public class AVAssetDownloadURLRequest: URLSession {
    func makeAssetDownloadTask(with URLAsset: AVURLAsset, assetTitle title: String, assetArtworkData artworkData: Data?, options: [String : AnyObject]? = [:])
        -> AVAssetDownloadTask?
}

public let AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: String
public let AVAssetDownloadTaskMediaSelectionKey: String
AVAssetDownloadTask
AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset
func setupAssetDownload() {
    let hlsAsset = AVURLAsset(url: assetURL)

    let backgroundConfiguration = URLSessionConfiguration.background(
        withIdentifier: "assetDownloadConfigurationIdentifier")

    let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
        assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

    // Download a Movie at 2 mbps
    let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
        assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!
    assetDownloadTask.resume()
}

func setupAssetDownload() {

    let hlsAsset = AVURLAsset(url: assetURL)

    let backgroundConfiguration = URLSessionConfiguration.background(
        withIdentifier: "assetDownloadConfigurationIdentifier")

    let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
        assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

    // Download a Movie at 2 mbps
    let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
        assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!
    assetDownloadTask.resume()
}

func setupAssetDownload() {
    let hlsAsset = AVURLAsset(url: assetURL)

    let backgroundConfiguration = URLSessionConfiguration.background(
        withIdentifier: "assetDownloadConfigurationIdentifier")

    let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
        assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

    // Download a Movie at 2 mbps
    let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
        assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!
    assetDownloadTask.resume()
}
// Setup and Start AVAssetDownloadTask

func setupAssetDownload() {
    let hlsAsset = AVURLAsset(url: assetURL)

    let backgroundConfiguration = URLSessionConfiguration.background(
        withIdentifier: "assetDownloadConfigurationIdentifier")

    let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
        assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

    // Download a Movie at 2 mbps
    let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
        assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!

    assetDownloadTask.resume()
}

func setupAssetDownload() {
    let hlsAsset = AVURLAsset(url: assetURL)

    let backgroundConfiguration = URLSessionConfiguration.background(
        withIdentifier: "assetDownloadConfigurationIdentifier")

    let assetURLSession = AVAssetDownloadURLSession(configuration: backgroundConfiguration,
        assetDownloadDelegate: self, delegateQueue: OperationQueue.main())

    let assetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: hlsAsset, assetTitle: "My Movie",
        assetArtworkData: nil, options: [AVAssetDownloadTaskMinimumRequiredMediaBitrateKey: 2000000])!

    assetDownloadTask.resume()
}
AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset
public protocol AVAssetDownloadDelegate: URLSessionTaskDelegate {

    optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
        didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue],
        timeRangeExpectedToLoad: CMTimeRange)

    optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
        didFinishDownloadingTo location: URL)
}

Monitoring the Download
AVAssetDownloadDelegate
Monitoring the Download

**AVAssetDownloadDelegate**

```swift
public protocol AVAssetDownloadDelegate: URLSessionTaskDelegate {
    optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
        didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue],
        timeRangeExpectedToLoad: CMTimeRange)

    optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
        didFinishDownloadingTo location: URL)
}
```
Monitoring the Download

AVAssetDownloadDelegate

```swift
public protocol AVAssetDownloadDelegate: URLSessionTaskDelegate {
    optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
                             didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue],
                             timeRangeExpectedToLoad: CMTimeRange)

    optional func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
                              didFinishDownloadingTo location: URL)
}
```
public protocol AVAssetDownloadDelegate: URLSessionTaskDelegate {
    
    optional func urlSession(_: URLSession, assetDownloadTask: AVAssetDownloadTask, didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue], timeRangeExpectedToLoad: CMTimeRange)

    
    optional func urlSession(_: URLSession, assetDownloadTask: AVAssetDownloadTask, didFinishDownloadingTo location: URL)
}

Monitoring the Download
AVAssetDownloadDelegate
// In-progress Delegate Methods

class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
    func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask,
                didLoad timeRange: CMTimeRange, totalTimeRangesLoaded loadedTimeRanges: [NSValue],
                  timeRangeExpectedToLoad: CMTimeRange) {
        // Convert loadedTimeRanges to CMTimeRanges
        var percentComplete = 0.0
        for value in loadedTimeRanges {
            let loadedTimeRange: CMTimeRange = value.timeRangeValue
            percentComplete += CMTimeGetSeconds(loadedTimeRange.duration) / 
            CMTimeGetSeconds(timeRangeExpectedToLoad.duration)
        }
        percentComplete *= 100
        print("percent complete: \(percentComplete)")
    }
}
class MyAppDelegate: UIResponder, UIApplicationDelegate {
    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]? = [:]) -> Bool {
        let configuration = URLSessionConfiguration.background(withIdentifier: "assetDownloadConfigurationIdentifier")
        let session = URLSession(configuration: configuration)
        session.getAllTasks { tasks in
            for task in tasks {
                if let assetDownloadTask = task as? AVAssetDownloadTask {
                    // restore progress indicators, state, etc...
                }
            }
        }
    }
}

// Restore Tasks on App Launch
// Restore Tasks on App Launch

class MyAppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]? = [:]) -> Bool {

        let configuration = URLSessionConfiguration.background(withIdentifier:
            "assetDownloadConfigurationIdentifier")
        let session = URLSession(configuration: configuration)

        session.getAllTasks { tasks in
            for task in tasks {
                if let assetDownloadTask = task as? AVAssetDownloadTask {
                    // restore progress indicators, state, etc...
                }
            }
        }
    }
}
// Restore Tasks on App Launch

class MyAppDelegate: UIResponder, UIApplicationDelegate {
    func application(_ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [NSObject : AnyObject]? = [:]) -> Bool {
        let configuration = URLSessionConfiguration.background(withIdentifier:
            "assetDownloadConfigurationIdentifier")
        let session = URLSession(configuration: configuration)

        session.getAllTasks { tasks in
            for task in tasks {
                if let assetDownloadTask = task as? AVAssetDownloadTask {
                    // restore progress indicators, state, etc...
                }
            }
        }
    }
}
AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset
// Store Location of Downloaded Asset

class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
    // called whenever anything is deposited at location
    func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask, didFinishDownloadingTo location: URL) {
        // Unlike URLSessionDownloadDelegate, Do NotMove Asset From This Location
        let locationToSave = location.relativePath!
        // Stash Away This Location
        ...
    }
}
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {

    // called whenever anything is deposited at location
    func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask, didFinishDownloadingTo location: URL) {
        // Unlike URLSessionDownloadDelegate, Do Not Move Asset From This Location
        let locationToSave = location.relativePath!
        // Stash Away This Location
        ...
    }
}
// Store Location of Downloaded Asset

class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
    // called whenever anything is deposited at location
    func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask, didFinishDownloadingTo location: URL) {
        // Unlike URLSessionDownloadDelegate, Do Not Move Asset From This Location
        let locationToSave = location.relativePath!
        // Stash Away This Location
        ...
    }
}
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
    // called whenever anything is deposited at location
    func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask, didFinishDownloadingTo location: URL) {
        // Unlike URLSessionDownloadDelegate, Do Not Move Asset From This Location
        let locationToSave = location.relativePath!
        // Stash Away This Location
        ...
    }
}
// Store Location of Downloaded Asset

class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {

    // called whenever anything is deposited at location
    func urlSession(_ session: URLSession, assetDownloadTask: AVAssetDownloadTask, didFinishDownloadingTo location: URL) {
        // Unlike URLSessionDownloadDelegate, Do Not Move Asset From This Location
        let locationToSave = location_relativePath!

        // Stash Away This Location

        ...
    }

}
AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
    func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: NSError?) {
        guard error == nil else { return }
        let assetURLSession = session as! AVAssetDownloadURLSession
        let assetDownloadTask = task as! AVAssetDownloadTask
        let audioGroup: AVMediaSelectionGroup = ...
        let spanishOption: AVMediaSelectionOption = ...
        guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as? AVMutableMediaSelection else { return }
        additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
        let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil, options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
        newAssetDownloadTask.resume()
    }
}
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {

func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError error: NSError?) {

    guard error == nil else { return }
    let assetURLSession = session as! AVAssetDownloadURLSession
    let assetDownloadTask = task as! AVAssetDownloadTask
    let audioGroup: AVMediaSelectionGroup = ...
    let spanishOption: AVMediaSelectionOption = ...

    guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as? AVMutableMediaSelection else { return }
    additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
    let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil, options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
    newAssetDownloadTask.resume()
}
}
// Download Additional Media Selections

class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {

    func urlSession(_ session: URLSession, task: URLSessionTask, 
didCompleteWithError error: NSError?) {
        guard error == nil else { return }
        let assetURLSession = session as! AVAssetDownloadURLSession
        let assetDownloadTask = task as! AVAssetDownloadTask

        let audioGroup: AVMediaSelectionGroup = ...
        let spanishOption: AVMediaSelectionOption = ...

        guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as? 
            AVMutableMediaSelection else { return }
        additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
        let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: 
            assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil, 
            options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
        newAssetDownloadTask.resume()
    }
}
// Download Additional Media Selections

class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {
    func urlSession(_ session: URLSession, task: URLSessionTask,
        didFinishDownloadingAtLocation location: URL?,
        withError error: NSError?) {
        guard error == nil else { return }
        let assetURLSession = session as! AVAssetDownloadURLSession
        let assetDownloadTask = task as! AVAssetDownloadTask
        let audioGroup: AVMediaSelectionGroup = ...
        let spanishOption: AVMediaSelectionOption = ...
        guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as?
            AVMutableMediaSelection else { return }
        additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
        let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset:
            assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil,
            options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!
        newAssetDownloadTask.resume()
    }
}
// Download Additional Media Selections

class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {

    func urlSession(_ session: URLSession, task: URLSessionTask,
        didCompleteWithError error: NSError?) {
        guard error == nil else { return }
        let assetURLSession = session as! AVAssetDownloadURLSession
        let assetDownloadTask = task as! AVAssetDownloadTask
        let audioGroup: AVMediaSelectionGroup = ...
        let spanishOption: AVMediaSelectionOption = ...

        guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as? AVMutableMediaSelection else { return }

        additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)

        let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil, options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])!

        newAssetDownloadTask.resume()
    }
}
class MyAssetDownloadDelegate: NSObject, AVAssetDownloadDelegate {

    func urlSession(_ session: URLSession, task: URLSessionTask, 
    didCompleteWithError error: NSError?) {
        guard error == nil else { return }
        let assetURLSession = session as! AVAssetDownloadURLSession
        let assetDownloadTask = task as! AVAssetDownloadTask
        let audioGroup: AVMediaSelectionGroup = ...
        let spanishOption: AVMediaSelectionOption = ...
        guard let additionalMediaSelection = self.downloadedMediaSelection?.mutableCopy() as? 
            AVMutableMediaSelection else { return }
        additionalMediaSelection.selectMediaOption(spanishOption, in: audioGroup)
        let newAssetDownloadTask = assetURLSession.makeAssetDownloadTask(asset: 
            assetDownloadTask.urlAsset, assetTitle: "My Movie", assetArtworkData: nil,
            options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])
        newAssetDownloadTask.resume()
    }
}
AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset
// Instantiating Your AVAsset for Playback

// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
    assetArtworkData: nil, options: nil)

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)
// Instantiating Your AVAsset for Playback

// 1) Create Asset for AVAssetDownloadTask

```swift
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
                                                      assetArtworkData: nil, options: nil)
```

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch

```swift
let playerItem = AVPlayerItem(asset: task.urlAsset)
```
// Instantiating Your AVAsset for Playback
// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
          assetArtworkData: nil, options: nil)

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)
One Week Later...
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
          assetArtworkData: nil, options: nil)

let playerItem = AVPlayerItem(asset: task.urlAsset)
// Instantiating Your AVAsset for Playback
// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
    assetArtworkData: nil, options: nil)

// 2) Re-use Asset for Playback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)

// 3) When Your Original AVURLAsset Instantiated with a Network URL is No Longer Available
let fileURL = URL(fileURLWithPath: self.savedAssetDownloadLocation)
let asset = AVURLAsset(url: fileURL)
let playerItem = AVPlayerItem(asset: task.urlAsset)

// 4) Augmenting a Download with Additional Media Selection
let task = session.makeAssetDownloadTask(asset: playerItem.asset as! AVURLAsset,
    assetTitle: "My Movie", assetArtworkData: nil, options: [AVAssetDownloadTaskMediaSelectionKey: additionalMediaSelection])
// Instantiating Your AVAsset for Playback

// 1) Create Asset for AVAssetDownloadTask
let networkURL = URL(string: "http://example.com/master.m3u8")!
let asset = AVURLAsset(url: networkURL)
let task = assetDownloadSession.makeAssetDownloadTask(asset: asset, assetTitle: "My Movie",
assetArtworkData: nil, options: nil)

// 2) Re-use Asset forPlayback, Even After Task Restoration at App Launch
let playerItem = AVPlayerItem(asset: task.urlAsset)

// 3) When Your Original AVURLAsset Instantiated with a Network URL is No Longer Available
let fileURL = URL(fileURLWithPath: self.savedAssetDownloadLocation)
let asset = AVURLAsset(url: fileURL)
let playerItem = AVPlayerItem(asset: task.urlAsset)

// 4) Augmenting a Download with Additional Media Selection
let task = session.makeAssetDownloadTask(asset: playerItem.asset as! AVURLAsset,
    assetTitle: "My Movie", assetArtworkData: nil, options: nil)
Query for Cached Media Selections
AVAssetCache

public class AVURLAsset {
    public var assetCache: AVAssetCache? { get }
}

public class AVAssetCache {
    public var isPlayableOffline: Bool { get }

    public func mediaSelectionOptions(in mediaSelectionGroup: AVMediaSelectionGroup) -> [AVMediaSelectionOption]
}

Query for Cached Media Selections

AVAssetCache

```swift
public class AVURLAsset {
    public var assetCache: AVAssetCache? { get }
}

public class AVAssetCache {
    public var isPlayableOffline: Bool { get }

    public func mediaSelectionOptions(in mediaSelectionGroup: AVMediaSelectionGroup) -> [AVMediaSelectionOption]
}
```
public class AVURLAsset {

    public var assetCache: AVAssetCache? { get }

}

public class AVAssetCache {

    public var isPlayableOffline: Bool { get }

    public func mediaSelectionOptions(in mediaSelectionGroup: AVMediaSelectionGroup) -> [AVMediaSelectionOption]

}
public class AVURLAsset {
    public var assetCache: AVAssetCache? { get }
}

public class AVAssetCache {
    public var isPlayableOffline: Bool { get }
}

public func mediaSelectionOptions(in mediaSelectionGroup: AVMediaSelectionGroup) -> [AVMediaSelectionOption]
AVAssetDownloadTask

1. Set up and start AVAssetDownloadTask
2. Monitor progress of download
3. Store location of downloaded asset
4. Download additional media selections
5. Play downloaded asset
Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS
Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS
AVFoundation handles packaging keys for offline storage
Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS
AVFoundation handles packaging keys for offline storage
App is expected to store its own keys to disk
Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS
AVFoundation handles packaging keys for offline storage
App is expected to store its own keys to disk
Support for offline keys is opt-in in the key server
Securing Your Offline Content

Offline FPS content protection

Same protections provided by online FPS apply to offline FPS
AVFoundation handles packaging keys for offline storage
App is expected to store its own keys to disk
Support for offline keys is opt-in in the key server
All offline FPS Keys must be declared as EXT-X-SESSION-KEYS
Offline FPS
Request flow

Your Key Server

Internet

Your App
AVFoundation Delegate

AVFoundation
Offline FPS
Request flow

1. Your app asks AVFoundation to download or play your protected HLS asset
AVFoundation will download your m3u8 playlist containing the KEY tag.
Offline FPS

Request flow

3. AVFoundation will call your app delegate to request the key
Offline FPS

Request flow

4. Your app delegate calls AVFoundation to create an FPS Server Playback Context request
5 Your app delegate sends the FPS SPC to your key server, which returns a FairPlay Content Key Context.
Offline FPS
Request flow

6. Your app delegate sends the CKC to AVFoundation to create a persistable CKC.
Offline FPS
Request flow

7. Your app writes the persistable CKC to your app storage

Diagram:
- Your Key Server
- Internet
- App Storage
- AVFoundation Delegate
- AVFoundation
- Your App
Offline FPS
Request flow

8. Your app provides the persistable CKC to AVFoundation
// FPS Key Fetch

func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
    shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
            contentIdentifier: contentID, options: nil)
        // send serverPlaybackContext to server to get contentKeyContext
        ...
        loadingRequest.dataRequest!.respond(with: contentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
// FPS Key Fetch

func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {

if loadingRequest.request.url?.scheme == "skd" {
    let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
        contentIdentifier: contentID, options: nil)
        
        // send serverPlaybackContext to server to get contentKeyContext
        
        ... 

        loadingRequest.dataRequest!.respond(with: contentKeyContext)
        loadingRequest.finishLoading()
        return true

    }

return false

}
// FPS Key Fetch
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
                                contentIdentifier: contentID, options: nil)
        // send serverPlaybackContext to server to get contentKeyContext
        ...
        loadingRequest.dataRequest!.respond(with: contentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
// FPS Key Fetch

func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
                              contentIdentifier: contentID, options: nil)
        // send serverPlaybackContext to server to get contentKeyContext
        ...

        loadingRequest.dataRequest!.respond(with: contentKeyContext)
        loadingRequest.finishLoading()

        return true
    }

    return false
}
public class AVAssetResourceLoadingRequest {
    public func persistentContentKey(fromKeyVendorResponse keyVendorResponse: Data,
                                      options: [String : AnyObject]? = [:], error outError: NSErrorPointer) -> Data
}

public let AVStreamingKeyDeliveryPersistentContentKeyType: String
public let AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: String
Offline FPS
Client changes

```swift
public class AVAssetResourceLoadingRequest {

  public func persistentContentKey(fromKeyVendorResponse keyVendorResponse: Data,
  options: [String : AnyObject]? = [:], error outError: NSErrorPointer) -> Data
}

public let AVStreamingKeyDeliveryPersistentContentKeyType: String

public let AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: String
```
public class AVAssetResourceLoadingRequest {
    public func persistentContentKey(fromKeyVendorResponse keyVendorResponse: Data,
                                       options: [String : AnyObject]? = [:], error outError: NSErrorPointer) -> Data
}

public let AVStreamingKeyDeliveryPersistentContentKeyType: String
public let AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: String
public class AVAssetResourceLoadingRequest {
    public func persistentContentKey(fromKeyVendorResponse keyVendorResponse: Data, options: [String : AnyObject]? = [:], error outError: NSErrorPointer) -> Data
}

public let AVStreamingKeyDeliveryPersistentContentKeyType: String

public let AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: String
// FPS Key Fetch for Persistent Keys

func resourceLoader(_ resourceLoader: AVAssetResourceLoader, 
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert, 
                                    contentIdentifier: contentID, 
                                    options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
        // send serverPlaybackContext to server to get contentKeyContext
        ...
        let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc, 
                                    options: nil, error: nil)
        persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
            contentIdentifier: contentID,
            options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])

        // send serverPlaybackContext to server to get contentKeyContext
        ...

        let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
            options: nil, error: nil)
        persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {

if loadingRequest.request.url?.scheme == "skd" {

let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
contentIdentifier: contentID,
options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])

// send serverPlaybackContext to server to get contentKeyContext
...

let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
options: nil, error: nil)

persistentContentKeyContext.write(to: keySaveLocation, atomically: true)

loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType

loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)

loadingRequest.finishLoading()

return true
}

return false
}
// FPS Key Fetch for Persistent Keys
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
            contentIdentifier: contentID,
            options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
        // send serverPlaybackContext to server to get contentKeyContext
        ...

        let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
            options: nil, error: nil)
        persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
            contentIdentifier: contentID,
            options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
        // send serverPlaybackContext to server to get contentKeyContext
        ...
        let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
            options: nil, error: nil)
        persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
            contentIdentifier: contentID,
            options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
        // send serverPlaybackContext to server to get contentKeyContext
        ...
        let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
            options: nil, error: nil)
        persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
// FPS Key Fetch for Persistent Keys

func resourceLoader(_ resourceLoader: AVAssetResourceLoader,
                      shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let serverPlaybackContext = loadingRequest.streamingContentKeyRequestData(forApp: appCert,
            contentIdentifier: contentID,
            options: [AVAssetResourceLoadingRequestStreamingContentKeyRequestRequiresPersistentKey: true])
        // send serverPlaybackContext to server to get contentKeyContext
        ...
        let persistentContentKeyContext = loadingRequest.persistentContentKey(fromKeyVendorResponse: ckc,
            options: nil, error: nil)
        persistentContentKeyContext.write(to: keySaveLocation, atomically: true)
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
// FPS Key Fetch for Persistent Keys

func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let persistentContentKeyContext = Data(contentsOf: keySaveLocation)!
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    } else {
        return false
    }
}
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource
loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let persistentContentKeyContext = Data(contentsOf: keySaveLocation)!
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    }
    return false
}
// FPS Key Fetch for Persistent Keys

func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource
  loadingRequest: AVAssetResourceLoadingRequest) -> Bool {

  if loadingRequest.request.url?.scheme == "skd" {
    let persistentContentKeyContext = Data(contentsOf: keySaveLocation)!

    loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
    loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
    loadingRequest.finishLoading()

    return true
  }

  return false
}
// FPS Key Fetch for Persistent Keys

func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    if loadingRequest.request.url?.scheme == "skd" {
        let persistentContentKeyContext = Data(contentsOf: keySaveLocation)!
        loadingRequest.contentInformationRequest!.contentType = AVStreamingKeyDeliveryPersistentContentKeyType
        loadingRequest.dataRequest!.respond(with: persistentContentKeyContext)
        loadingRequest.finishLoading()
        return true
    } else {
        return false
    }
}
Asset Management
Best practices
Asset Management

Best practices

Clean up unneeded assets on disk
Asset Management

Best practices

Clean up unneeded assets on disk

• Cancelled downloads remain on disk
Asset Management

Best practices

Clean up unneeded assets on disk

• Cancelled downloads remain on disk

Downloads should be driven by explicit user actions
Asset Management

Best practices

Clean up unneeded assets on disk

• Cancelled downloads remain on disk

Downloads should be driven by explicit user actions

Downloads are opted out of iCloud backup
Asset Management

Best practices

Clean up unneeded assets on disk
- Cancelled downloads remain on disk

Downloads should be driven by explicit user actions

Downloads are opted out of iCloud backup

Be prepared for the system to delete your assets to reclaim disk space
Asset Management

Best practices

Clean up unneeded assets on disk
• Cancelled downloads remain on disk

Downloads should be driven by explicit user actions

Downloads are opted out of iCloud backup

Be prepared for the system to delete your assets to reclaim disk space

Keep downloaded assets at the system-provided location
Asset Management

Best practices

Clean up unneeded assets on disk
  • Cancelled downloads remain on disk

Downloads should be driven by explicit user actions

Downloads are opted out of iCloud backup

Be prepared for the system to delete your assets to reclaim disk space

Keep downloaded assets at the system-provided location

If server asset changes, host the modified asset at a new URL
Summary
Summary

MPEG-4 Fragment support
Summary

MPEG-4 Fragment support

• Supports cross-ecosystem interoperability
Summary

MPEG-4 Fragment support

• Supports cross-ecosystem interoperability
• Compatible with all HLS features
Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists
Summary

MPEG-4 Fragment support
• Supports cross-ecosystem interoperability
• Compatible with all HLS features
• Minimal changes to HLS playlists

In-playlist metadata
MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

- `#EXT-X-DATERANGE`
Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

- `#EXT-X-DATERANGE`
- Great for live content with updating metadata
Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

- `#EXT-X-DATERANGE`
- Great for live content with updating metadata

Offline HLS playback
Summary

MPEG-4 Fragment support
• Supports cross-ecosystem interoperability
• Compatible with all HLS features
• Minimal changes to HLS playlists

In-playlist metadata
• \#EXT-X-DATERANGE
• Great for live content with updating metadata

Offline HLS playback
• Configurable media downloading
Summary

MPEG-4 Fragment support

- Supports cross-ecosystem interoperability
- Compatible with all HLS features
- Minimal changes to HLS playlists

In-playlist metadata

- `#EXT-X-DATERANGE`
- Great for live content with updating metadata

Offline HLS playback

- Configurable media downloading
- Industrial strength content protection
More Information

https://developer.apple.com/wwdc16/504
# Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advances in AVFoundation Playback</td>
<td>Mission</td>
<td>Wednesday 9:00AM</td>
</tr>
<tr>
<td>AVKit on tvOS</td>
<td>Presidio</td>
<td>Friday 11:00AM</td>
</tr>
<tr>
<td>HTTP Live Streaming Authoring and Validation</td>
<td>Video</td>
<td>Watch on Demand</td>
</tr>
<tr>
<td>Labs</td>
<td>Graphics, Games, and Media Lab C</td>
<td>Time</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>HTTP Live Streaming Lab</td>
<td></td>
<td>Wednesday 4:00PM</td>
</tr>
<tr>
<td>AVFoundation / HTTP Live Streaming Lab</td>
<td></td>
<td>Thursday 9:00AM</td>
</tr>
<tr>
<td>AVKit Lab</td>
<td></td>
<td>Friday 1:00PM</td>
</tr>
</tbody>
</table>